アルゴリズム 7－4
（1989. 5. 22）

最適ハッシュ関数の幾何学的構成法

浅野哲夫[1]　　徳山豪[2]
[1]大阪電気通信大学　　[2]日本ＩＢＭ　東京基礎研究所

本論文では、n 個のキーの集合とそれらのキーに対して定義される任意の関数 h_1, h_2 が与えられたとき、これらの関数の1次結合によって、ある基準の下に最適なハッシュ関数を構成するアルゴリズムを提案する。本文で考えるハッシュ関数は、最大値と最小値の間を定数個のバケットと呼ばれる区間に分割したとき、その区間名を関数値とするものである。このとき、1つのバケットに入るキーの個数の最大値を最小にすることや、空のバケットの個数を最小にすることが要求される。本論文ではそのような基準の下で最適なハッシュ関数を関数 h_1 と h_2 の1次結合の形で求めるアルゴリズムを3種類提案する。これらは現在知られている最も効率のよいアルゴリズムを計算時間と記憶量の両面で改善したものである。

A Geometric Construction of Optimal Hash Function

Tetsuo Asano[1] and Takeshi Tokuyama[2]
[1]Osaka Electro-Communication University
[2]IBM Research, Tokyo Research Laboratory

ABSTRACT

Given a fixed set S of keys and two arbitrary functions h_1 and h_2 associated with each key, we want to find a hash function by linear combination of those functions that is optimal in some criterion. Various criteria may be considered, such as minimization of the maximum number of keys to be included in one bucket or minimization of the number of empty buckets. This paper presents three different efficient algorithms for finding such a hash function based on dual transform between points and lines in the plane. They improved the previous best result both in time and space complexities.

1. Introduction

Given a static set of keys, it is not so easy to devise a perfect hash function. When two quantities are easily computable for each key, we can define various hash functions by linear combination of those quantities. In this paper we are interested in the problem of finding hash functions that are optimal in several criteria. When the hash table size is fixed, one of the criteira is to minimize the maximum number of keys associated with the same bucket and the other is to minimize the number of empty buckets (those to which no key is assigned).

Since two quantities are available for each key, we may regard a set of keys as that of points in the plane. When we project each point onto a line of angle θ that passes through the origin, the distance from the origin to the projective point of a point (x_i, y_i) is expressed as $z_i(\theta) = x_i \cos\theta + y_i \sin\theta$, which is a linear combination of two coordinates. When the hash table size is fixed, say b, one natural way of defining buckets is to divide equally the interval on the projective line between the leftmost and rightmost projected points. The problem to be considered in this paper is to find an angle to optimize the distribution of projective points in several different criteria. We further show that the original problem that is to devise an optimal hash function using linear combination of two functions is reduced to solve the above-stated problem.

It is easy to show that the above problem is equivalent to the following geometric problem:

Given a set S of n points in the plane and an angle θ, a set of $b+1$ lines $l_0, l_1, \ldots, l_b$ of slope θ (b is fixed) is called a θ-cut of S if l_0 and l_b are two supporting lines of S and $l_0, l_1, \ldots, l_b$ are equally spaced. The regions between two consecutive lines are called buckets. Then, find an optimal θ-cut such that points are grouped into b buckets most uniformly, in other words, such that the maximum number of points to be included in one bucket is minimized.

This problem was first presented by Sprugnoli [Sp77]. Comer and O'Donnell [CO82] proposed an efficient algorithm for this problem which runs in $O(bn^2 \log nb)$ time using $O(n^2 + bn)$ space. In this paper we present two different linear-space algorithms, based on duality transformation [CGL83]. They first construct an arrangement of n lines dual to given points and partition it into slabs at the vertices of the upper and lower envelopes. These slabs are further decomposed into b trapezoids by $b-1$ equally spaced lines (called *bucket lines*), which are dual to boundaries between adjacent buckets. Then, the optimal angle can be found by enumerating all the intersections between the dual lines and trapezoids. The first algorithm (Algorithm 2) runs in $O(n^2 + K \log n + bn)$ time, and is based on Bentley and Ottman's intersection reporting algorithm [BO79], where K is the number of intersections reported.

The other algorithm (Algorithm 3) is advantageous if $b < \sqrt{n}$. It performs a simplex range search [CGL83, Ch87, Ed87, EW86, We88, Wi82] in each slab to enumerate all the lines that intersect *bucket lines*, and runs in $O(b^{0.610}n^{1.695} + bn + K \log n)$ time.

The dominant term in the above time complexities may be $K \log n$, although K is expected to be relatively small. We prove that K, the number of intersections between n lines dual to given points and bucket lines is $O(bn+n^2)$ for the worst case. Thus, our algorithms are superior to Comer and O'Donnell's algorithm both in time and space since their algorithm requires $O(bn^2 \log bn)$ time and $O(n^2 + bn)$ space.

Another point emphasized in this paper is a new method for analyzing the number of intersections between line segments, which is based on the property of envelopes of an arrangement, and totally different from the method based on the Davenport-Schinzel sequence [DS65, HS86].

2. Naive Plane Sweep Method

Our methods are based on the point-to-line geometric transform [CGL83, Ed87]. A point set is transformed into an arrangement of lines. A line of an angle θ is mapped to a point on a vertical line $x = \tan\theta$. We should sweep on θ in order to solve our problem. The advantage of considering the dual problem is that we can transform a circular sweep into an ordinary plane sweep.

Let S be a set of n points in the plane. For an angle θ, $-\pi/2 < \theta < \pi/2$, let $l_0(\theta)$ and $l_b(\theta)$ be two lines at angle θ supporting S. The region between them is divided by $b-1$ equally spaced lines $l_1(\theta), \ldots, l_{b-1}(\theta)$ (see Fig. 1). Such a set of lines is called a θ-cut of S. The region bounded by $l_i(\theta)$ and $l_{i+1}(\theta)$ is called the i-th bucket, denoted by $B(\theta, i)$. The cardinality of $B(\theta, i)$ is denoted by $\beta(\theta, i)$, or $\beta(i)$ if we regard θ as a parameter. Throughout the paper the number b is fixed. We denote the geometric transform by D. Thus, $D(l_0(\theta)), \ldots, D(l_b(\theta))$ lie on the the vertical line $x = \tan(\theta)$; moreover, they divide the interval $[D(l_0(\theta)), D(l_b(\theta))]$ evenly. The locus of $D(l_0(\theta))$ ($D(l_b(\theta))$, resp.) over the parameter θ forms the upper envelope (lower envelope, resp.) of the arrangement of lines dual to S. Therefore, that of $D(l_i(\theta))$ forms a chain of line segments (called *the i-th bucket line*) as shown in Fig. 2. For an angle θ, a point of S is contained in the i-th bucket $B(\theta, i)$ if and only if its dual line intersects the vertical interval $[D(l_i(\theta)), D(l_{i+1}(\theta))]$.

The algorithms to be presented in this paper are based on the same preprocessing. We first construct the upper and lower envelopes and the bucket lines. Then, we partition the region bounded by those envelopes by vertical lines at every vertex on the envelopes. This results in $O(n)$ slabs each with two vertical sides (except the leftmost and rightmost ones). Next, we decompose each slab by

bucket lines, as shown in Fig. 3. This preprocessing takes $O(n \log n + nb)$ time.

Algorithm 1: Naive Plane Sweep

(1) We have n lines and $O(b)$ bucket lines. Enumerate all the intersections by using the intersection reporting algorithm developed by Bentley and Ottman [BO79]. Let K be the total number of intersections.

(2) Sort those intersections in one direction.

(3) Based on the sorted list obtained, perform a plane sweep. At each intersection with the ith bucket line we must decrease $\beta(i-1)$ and increase $\beta(i)$, or to increase $\beta(i-1)$ and decrease $\beta(i)$, depending on the slope of the line giving the intersection. We must also compare the distribution obtained with the optimal distribution found thus far.

There are various criteria on the cost of distribution of points into buckets. The time for evaluating the cost, given b buckets and updating the cost function at each step of the plane sweep, denoted by $T(b)$ and $U(b)$, respectively, depends on the criterion selected. When we want to minimize the maximum bucket size, we can achieve $T(b) = O(b)$ and $U(b) = O(1)$ by using a relatively simple data structure. The space needed for this data structure is $O(b)$. On the other hand, if we are interested in minimizing the variance of cardinalities of buckets, we have only to keep the number of empty buckets. Thus, a simpler data structure suffices.

[Theorem 1] Algorithm 1 finds an optimal cut giving the optimal distribution of points in $O((bn + n^2 + K)\log n + K \log K + KU(b) + T(b))$ time and $O(bn + K)$ space.

Proof: The correctness of the algorithm follows from the above considerations. The space complexity is straightforward. The time complexity is obtained as follows. In steps 1 and 2, bucket lines are obtained in $O(n \log n + bn)$ time. Then, all the intersections are found in $O((bn + n^2 + K)\log n)$ time by Bentley and Ottman's algorithm. Sorting in Step (2) takes $O(K \log K)$ time. Lastly, in step (3), the distribution for a = negative infinity is evaluated. This requires $T(b)$ time. Then at each intersection we update the bucket count in $U(b)$ time to evaluate the resulting distribution. Thus, we have the time complexity in the theorem. □

3. Linear-Space Implementation

In Algorithm 1 we applied Bentley and Ottman's algorithm to a set of lines and bn bucket lines. In the second algorithm within each slab we first find all the lines intersecting any bucket line. Once

we have such a set of lines, we can dynamically produce all the intersections in increasing order of their abscissa. To find such lines, we divide the left side of the slab into b segments of the same length. Then, for each line we can easily locate the intersections between the line and the left and right sides of the slab. Enumerating intersections in a smart way, we design a better algorithm.

Algorithm 2: Linear-Space Implementation

[1] The same preprocessing as for Algorithm 1 is performed.

[2] For each slab S_i from left to right do the following:

[2.1] Partition the slab S_i into b trapezoids with $b-1$ bucket lines.

[2.2] For each line determine whether it intersects any bucket line.
If it does, then put the first (leftmost) intersection into a heap.

[2.3] Perform a plane sweep as follows.

[2.3.1] Evaluate the distribution of lines into buckets at the left side of the slab.

[2.3.2] while(heap is not empty) {

[2.3.2.1] Extract the leftmost intersection (s_j, B_k) from the heap,
where s_j is a line and B_k is a bucket line.

[2.3.2.2] Update the distribution and put the next intersection given by
the line s_j into the heap if it lies in S_i.

[2.3.2.3] If the resulting distribution is better than the current optimum,
then update the current optimum.}

[Theorem 2] Algorithm 2 finds an optimal cut giving the optimal distribution of points in $O(n^2 + bn + K \log n + KU(b) + T(b))$ time and $O(n+b)$ space.

Proof: We shall concentrate on the time complexity of the algorithm. In the algorithm we perform a plane sweep slab by slab. For each slab S_i steps 2.1 and 2.2 can be done in $O(b)$ time and $O(n)$ time, respectively. The plane sweep of step 2.3 can be carried out in $O(k_i \log n + k_i U(b) + T(b))$ time, where k_i is the number of intersections to be processed in the slab. The time for evaluating the distribution at the left side can be omitted except for the leftmost slab, since the distribution at the left side of an intermediate slab is exactly the same as the one at the right side of the preceding slab. Summing up the time for each slab, we obtain the time complexity in the theorem. □

As is easily seen, Algorithm 2 is superior to Algorithm 1 both in time and space complexity.

4. Range-Search Based Algorithm

While Algorithm 2 is an improvement of Algorithm 1, it still requires time proportional to n^2, even if there are few intersections between lines and bucket lines. When there are few such intersections and b, the number of buckets is much smaller than n, and the algorithm to be described later may be useful. The above stated situation arises, for example, when we want to divide an enormous number of keys into a fixed number of blocks in a disk.

The most important operation in the algorithms described thus far is to enumerate all the intersections between bucket lines and infinite lines corresponding to given points. Considering the problem in its dual plane, it becomes to enumerate all the points contained in double wedges, since bucket lines are transformed into double wedges and infinite lines into points. This is a so-called triangular range-searching problem. It is known that any such range query can be answered in $O(n^{0.695}+k)$ time by using a Ham-Sandwich tree data structure after $O(n \log n)$-time preprocessing [CGL83, Ch87, Ed87, EW86, We88, Wi82]. The remaining operations are the same as in Algorithm 2. Thus, at each slab we can enumerate all the lines that intersect any bucket line in $O(bn^{0.695}+k_i)$ time by repeating a triangular range search b times. This leads to an algorithm that runs in $O(bn^{1.695}+K\log n+KU(b)+T(b))$ time and $O(n+b)$ space.

We shall show below that if $b < \sqrt{n}$, then the algorithm can be improved so that it runs in $O(b^{0.610}n^{1.695}+K\log n+KU(b)+T(b))$ time using linear space. The key idea is a combination of the bucketing technique and the erasing subdivision by Ham-Sandwich cuts [Ed87].

Let P be the convex hull of S. For an angle τ, $R(\tau)$ denotes a rectangle, with sides at angle τ, that circumscribes P. Let $R(\lambda)$ be such a rectangle with a minimal area. Then,

[Lemma 1] The area of $R(\lambda)$ is no more than twice of the area of P.

We cut $R(\lambda)$ with two sets of equally spaced $b-1$ lines at angles of λ and $\lambda+\pi/2$, respectively (see Fig. 4). Then, we have b^2 cells congruent to each other.

[Lemma 2] For any angle θ, a cell is cut with at most two lines, parallel cuts.

We store the points of $S \cap A$ for a cell A using the erasing subdivision by Ham-Sandwich cuts.

[Lemma 3] The number of intersections of the edges of the erasing subdivision in a cell with parallel cuts is $O(k^{0.695})$ for any angle, where k is the number of points in the cell.

We consider an efficient data structure to query the table $B(\theta)=(\beta(\theta,i))_{i=1,\ldots,b}$ for an arbitrary θ. We construct a quad tree of size b^2 on the cells. Each leaf of the tree points to the Ham-Sandwich tree associated with the erasing subdivision of the corresponding cell. Then,

[Proposition] Using the above data structure, we can query the table $B(\theta)$ in $O(n^{0.695}b^{0.610})$ if

$b < \sqrt{n}$.

Proof. The stabbing number of our tree by all lines of a parallel cut is

$$b^2 + \sum_{i=1}^{b^2} ck_i^{0.695} \cdots (1) \qquad (c : \text{a constant})$$

where k_i is the number of points in the i-th cell. The maximum of (1) is attained if the points are distributed evenly. Thus, the stabbing number is not greater than $b^2 + cb^2(\frac{n}{b^2})^{0.695} = b^2 + cb^{0.610}n^{0.695}$ Since $b^2 < b^{0.610}n^{0.695}$, if $b < \sqrt{n}$, we obtain the proposition.

Using the above data structure we can enumerate all the dual lines that intersect any bucket lines in $O(b^{0.610}n^{0.695} + k_i)$ time for each slab in the dual plane, where k_i is the number of lines reported. This leads to the following theorem:

[Theorem 3] Using the above data structure, an optimal cut is found in $O(b^{0.610}n^{1.695} + K \log n + KU(b) + T(b))$ time using $O(n + b)$ space.

The leading term in the time complexities of above-described algorithms may be $K \log n$. Although there seem to be $O(bn^2)$ intersections, we can prove that there are only $O(bn + n^2)$ intersections.

[Theorem 4] Given a set S of n points in the plane and the number b of buckets, there are at most $O(bn + n^2)$ intersections between the dual lines and bucket lines.

Proof: We can prove the theorem based on precise analysis of the number of intersections between a line and bucket lines. Although it is not trivial and is interesting itself, we omit the proof because of the lack of the space.

5. Conclusions

In this paper we have presented two different algorithms for finding the minimum cost distribution of projected points into buckets. They offer improvements over the existing best algorithm with respect to both time and space complexities. We could consider a similar problem: Given a line l and the screen l_0 parallel to l. A camera moves on the line l and reflect planar objects on the screen. Given a set of points in the plane, find the best camera position that generates the grouping of the minimum variance of number of points of groups. We can reduce the problem to our problem in this paper by using a projective transformation.

REFERENCES

[BO79] J.L. Bentley and T. Ottman: *Algorithms for Reporting and Counting Geometric Intersections*, IEEE

Trans. Comput., vol. C-28, pp. 643-647, Sept. 1979.
[CGL83] B. Chazelle, L.J. Guibas, and D.T. Lee: *The Power of Geometric Duality*, Proc. 24th Symposium on Foundations of Computer Science, pp. 217-225, 1983.
[Ch87] B. Chazelle: *Polytope Range Searching and Integral Geometry*, Proc. 28th Symposium on Foundations of Computer Science, pp. 1-10, 1987.
[Co82] D. Comer and M.J. O'Donnell: *Geometric Problems with Applications to Hashing*, SIAM J. Comput., vol. 11, No. 2, pp. 217-26, 1982.
[DS65] H. Davenport and A. Schinzel: *A Combinatorial Problem Connected with Differential Equations*, American Journal of Mathematics, vol. 87, pp. 684-694, 1965.
[Ed87] H. Edelsbrunner: *Algorithms in Combinatorial Geometry*, EATCS Monographs in Theoretical Computer Science, vol. 10, Springer-Verlag Berlin Heidelberg, 1987.
[EW86] H. Edelsbrunner and E. Welzl: *Half Planar Range Search in Linear Space and $O(n^{0.695})$ Query Time*, Inform. Process. Lett. vol. 23, pp. 289-193, 1986.
[HS86] S. Hart and M. Sharir: *Nonlinearity of Davenport-Schinzel Sequences and Generalized Path Compression Schemes*, Combinatorica, vol. 6, pp. 151-177, 1986.
[Sp77] R. Sprugnoli: *Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets*, Comm. ACM, vol. 20, pp. 841-850, 1977.
[To83] G.T. Toussaint: *Solving Geometric Problems with the Rotating Calipers*, in Proc. IEEE MELECON '83, Athens, Greece, May 1983.
[We88] E. Welzl: Partition Trees for Triangle Counting and Other Range Searching Problems, Proc. 4th Annual Symposium on Computational Geometry, pp. 23-33, Urbana-Champain, IL, June 1988.
[Wi82] D. Willard: *Polygon Retrieval*, SIAM J. Comput., vol. 11, pp. 149-165, 1982.

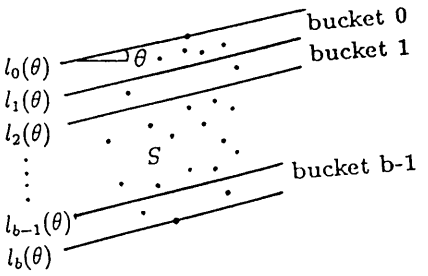

Fig. 1. A θ–cut of a point set S into b buckets.

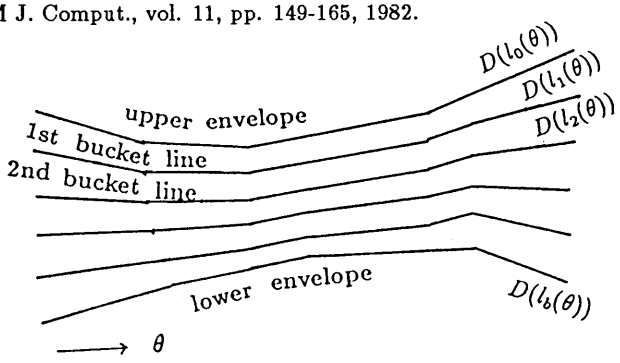

Fig. 2. Bucket lines in the dual plane.

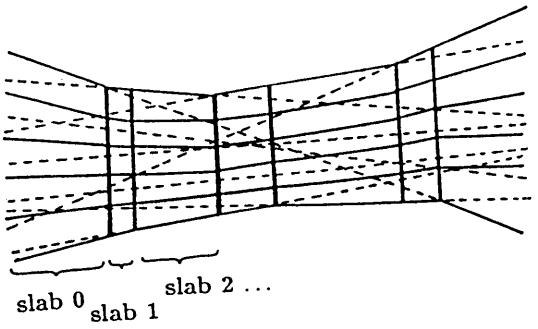

Fig. 3. Decomposition into slabs. Dotted lines correspond to given points.

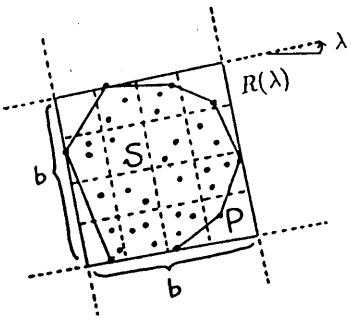

Fig. 4. Minimum enclosing rectangle $R(\lambda)$ and its decomposition into b^2 bucket.